Volume 2 No. 5, May 2012 ISSN 2223-4985

International Journal of Information and Communication Technology Research

©2012 ICT Journal. All rights reserved

http://www.esjournals.org

# Framework for Security and Integration for GUI-Based Operating System

**Er. Ritika Pandhi**
Student of M.Tech(CSE)
D.V.I.E.T, INDIA

## ABSTRACT

Although data mining is a relatively new term, not the technology. Companies have used powerful computers for analyze market research reports for years. However, it has new innovations while computer processing power, statistical software are dramatically increasing the accuracy of analysis and risk of security while droping cost. Generally, data or knowledge discovery is the process of analyzing data and patterns from many different dimensions or angles from different sources standardization it into useful information, that can be used to increase revenue .The various integrated tools introduced are pieces of software that are run on a particular dataset at a particular time for a particular purpose. In our paper we introduce a framework for security and integration for GUI based operation system such as Linux or Unix ,in which we allow various unexpected discoveries. We introduced security icons which work effectively with GUI which increase the utility in determining patterns , accuracy ,privacy and decreasing security issues.

**Keywords:** *GUI, LINUX, Gnome, KDE*

## 1. INTRODUCTION

Current mainstream operating systems that are used for desktop computing, e.g. Windows, Mac OS X, or Linux-based systems, proved unsatisfactory when it comes to security. The major problem with current systems is their inability to provide effective isolation between various programs running on one machine. E.g. if the users Web browser gets compromised, the OS is usually unable to protect other users applications and data from also being compromised. The above situation is a direct result of certain architectural design decisions, which include over-complexity of the OS API, insecure GUI design, the monolithic kernel architecture. We can try to take the reactive approach, as many vendors do today, and try to patch every single known security bug. But such approach not only scale well, but simply does not work .Patching can only address known and popular attacks, but offers no protection against new, or less popular threats. A GUI uses windows, icons, and menus to carry out commands such as opening files, deleting files, moving files and although many GUI OS are operated by mouse, keyboard. Unlike a command line OS like Unix or MS-DOS, GUI OS are much easier for end-users to learn and use because commands do not need to be known or memorized. Because of their ease of use, GUI OS have become the dominant OS used by end-users. Examples of a GUI OS are Microsoft Windows, Apple System 7 and Mac OS X etc. A few examples of a GUI interfaces are GNOME and KDE. Various aspects of secure GUI based operating systems have been analyzed as security patterns. However, existing patterns do not cover explicitly the secure interaction of users with the user interface of applications. Especially GUI tend to get complex and vulnerable to security measures and Malware such as Trojan Horses, spoofing and eavesdropping, and spyware which remain to be persistent security threats that exploit the overly complex GUI of today's commodity operating systems.. A secure graphical interface system has to provide a trusted path between the user and the application. The trusted path must be allow integrity, authentication, confidentiality of the transmitted data. We present a framework for security and integration for GUI systems and evaluate its use in different implementations which shows how to fulfill the security requirements of a trusted path while preserving, in a policy-driven way, the flexibility that graphical user interfaces generally demand. **Virtualization enable security?** Virtualization allows to create isolated containers, the Virtual Machines (VM). VMs can be much better isolated between each other than standard processes in monolithic kernels of popular OS like Windows or Linux. This is because the interface between the VM and the hypervisor can be much simpler, and also the hypervisor itself can be much simpler. Additionally modern computers have hardware support for virtualization which allows for further simplification of the hypervisor code, as well as for creating more robust system configurations, e.g. using so called driver domains special hardware isolated containers for hosting. The vast majority of visualization tools introduced so far are specialized pieces of software that are explicitly run on a particular dataset at a particular time for a particular

International Journal of Information and Communication Technology Research

©2012 ICT Journal. All rights reserved

http://www.esjournals.org

purpose. Although there are techniques to identify applications, such as verification of digital signatures of program binaries or secure bootstrap mechanisms, these identities need to be reliably shown to the user. For security-critical applications users need to be able to clearly identify the application they are interacting with, and to be sure the data they enter cannot be accessed by unauthorized applications. The user interface system must be able to isolate the input and output channels of different applications. In former times, applications behaved friendly and cooperated with each other to please the user. GUIs designed twenty five years ago do everything to facilitate communication among applications without bothering application programmers with security protocols. Today, computers are ubiquitously connected to networks and it is usual practice to download applications and applets from untrusted sources. This practice brings along the risk of falling prey to malware . Once installed on a system, malicious code can furthermore exploit the networking facilities to communicate. In 2008, the overdue discussion about what OS designers can do about GUI security was brought back to life by J. Shapiro with the presentation of the Trusted Window System. However , the presented approach supports only a dedicated set of applications compiled for the particular platform while the broad range of mass-market software is not available.

In contrast, current OS and virtualization techniques provide powerful ways of securing applications while retaining support for legacy software. It provides coarse grained isolation of concurrently running virtual machines and heads toward multi-level security. The isolation mechanisms implemented by these platforms enable the safe execution of security applications aside of a sandboxed legacy OS. With our work, we complement these techniques by an extremely minimized secure GUI to fight security problems such as spyware and Trojan Horses. We address the following classes of malware:

- **Spying on other applications**: No application should be able to obtain sensitive information from another application via the GUI, for ex. by taking a screen shot.
- **Spy which observing the user:** Spyware should not be able to sample key strokes and mouse actions.

**Denial of service of the user interface:** Current overly complex GUIs are prone to denial-of-service attacks driven by client applications. Such applications can infinitely grab the mouse pointer or open a full-screen window that captures all input events to make the user interface inaccessible. When relying on high availability, malicious or faulty applications are an unbearable risk. Trojan Horses mimic trusted applications to wrest sensitive information from the user. High complexity of system software implies a high risk for the robustness and security of dependent software components. The GUI server is a crucial part of system software on which end-user applications absolutely need to rely. Consequently, the design of the GUI server described in this section follows the principle of minimal complexity. We only integrate mechanisms for enforcing security and a core of mandatory GUI functions that cannot be implemented in another system component. However, the realization of a secure GUI system is an important. However, existing secure GUI systems have some elements in common on a conceptual level. We present a pattern for secure GUI systems and evaluate its use in the different Spyware and Trojan Horses are crucial security problems of today but they are not widely addressed by developers of OS and GUI, who are afraid of breaking compatibility with existing commodity applications.

## 2. SECURE GUI SYSTEM PATTERN

Intent: Provide a trusted path between the user and the application the user intends to use. Security-critical applications require integrity and confidentiality of user input and displayed output.

**(i)** Let us consider an **Example**:Consider a login dialog box on a graphical window system where the user has to enter a password. The user must be ensure that the dialog displayed corresponds to the login. Additionally, the data transmitted between the login dialog and the user should not be possible to be spyware and malwares .

**(ii)** **Related Situation**: You need to provide a GUI for various applications with different security requirements for one or more users. You need to ensure that the application the user in front of the input/output devices is currently interacting with, cannot appear to fake and cannot be appear for marware , eavesdrop or manipulated by other applications.

**(iii)** **Problem Related**: Providing security for user interfaces is non-trivial because they often rely on and can communicate with various other components. The problem is to combine the individual goals(security, authentica- tion and confidentiality and usability) in so that efficient interaction with the user interface remains possible. Applications can impose other applications by adapting the look and feel. An

International Journal of Information and Communication Technology Research

©2012 ICT Journal. All rights reserved

http://www.esjournals.org

attack on the user's privacy can be performed by applications that are marware, eavesdrop e.g., keyloggers typically install themselves as device drivers to intercept all keyboard events. In particular, the solution to this problem has to resolve the following forces:

- To provide a GUI system applications that have proper authentication, constrai- nts that have trusted path to invoke interaction at any time and can be used by several applications simultaneously. Unauthorized applications should not be able to eavesdrop. For example, in a multi-level security system copy & paste has to adhere to the information flow policy.

**(iv)** **Solution:** The central idea is to protect all user input/output through a Secure User Interface SUI system, and to separate the content drawn by applications from what is actually displayed on the screen. The SUI controls solely the input events from the user input devices and graphics rendering hardware. Only one application is active at a certain time, i.e., it has the "input focus". Hence, only the SUI decides how it is going to be displayed on the screen and what will be displayed and SUI has to detect trusted path invocation e.g., when the user wants to switch the input focus to another application.

**(v) Consequences for the Problem:**

Applying this pattern is expected to result in the following benefits:

- Trusted path for the user input and the application output is made. This channel is secure against marwares, eavesdrop manipulation by unauthorized applica- tions.
- The resulting SUI system is very flexible since labeling and access decisions can be delegated to the policy.

However, this pattern may also have the liabilities as consequences:

- The SUI might become a single point of failure. If the SUI does not work correctly, the whole system might become unusable for users, or the security of user intentions is compromised.
- There might be usability issues due to security constraints of GUI system. For example, when applying a window labeling policy, users might need extra training and education to understand the meaning of the colors.
- There is need to trust the SUI, the components for a high assurance system, it require high assurance for their development and integrity, eavesdrop-ping protection during runtime.

The actual implementation of the labeling in the SUI can be fitted to many scenarios. For example, one can implement different labeling methods for color blind people, and another can use a special braille external interface with separated "secure label zone" for blind people. It is also possible to decouple the label from the screen at all. For example, an additional small display attached to the keyboard could display the security level of the application that has currently the input focus. Another area of extension is to control audio input/output in the same way as it is done with keyboard inputs.

**(vi) Related Framework:**

On an architectural view, the Secure GUI System framework is a Single Access Point for users to the graphical interfaces of the system's applications, especially keyboard/mouse and processing unit. This is similar to the Secure GUI System pattern since the SUI system controls the usage of the graphics output and the user input and provide authenticity of the end points. Actually the Secure GUI System can use the Reference Monitor pattern to implement a policy-driven secure GUI. For example, digital certificates of program binaries can be used to reliably identify applications, which OS has to verify when a process is started. The SUI has to be executed in an Execution Domain which restricts the usage of the user I/O devices solely to the SUI. To protect the input sent to an application and the output sent to the graphics hardware, it must be possible to isolate memory. Process isolation can be achieved with the Virtual Address Space pattern. To protect the SUI process from other processes and to allow some applications additional privileges, the Secure Process pattern used to isolate processes and assign authorization rights. Some applications have special dedicated meaning and hence have additional privileges. For example, a screen snapshot application would need to capture the graphical output of other applications.

## III. CONCLUSION

Secure operating systems need secure user interfaces provide users a trusted path to applications. We

International Journal of Information and Communication Technology Research

©2012 ICT Journal. All rights reserved

http://www.esjournals.org

have presented a security pattern for graphical user interface systems. It shows how to control the access to user input/output devices such as keyboard, mouse, and display, and how to provide an indication of authenticity of applications to the user. The framework can be found in both research prototypes and commercial products. We expect this framework to be an important amendment to other operating system security patterns.

## REFERENCES

[1] Daniel A. Keim, Hans-Peter Kriegel, and Mihacl Ankerst. Recursive pattern: A technique for visualizing very large amounts of data. In Proc of IEEE Conference Visualization '2005'

[2] Wojciech Basalaj. Proximity visualization of abstract data. PhD thesis, University of Cambridge Computer Laboratory, 2000.

[3] Jeff Beddow Shape coding for multidimensional data on a microcomputer display. In Proceedings of IEEE Conference Visualization '00'.

[4] Herman Chernoff The use of faces to represent points in k-dimensional space graphically.

[5] J. Yoder and J. Barcalow , "Architectural patterns for enabling application security," in PLoP'97: Conference on Pattern Languages of Programs, 2007.

[6] Fabrizio Ferrandina , Thorsten Meyer, and Roberto Zicari. Implementing lazy database updates for an object database system. In Proceedings of the Twentieth International Conference on Very Large Databases, 2004.

[7] John P. Lewis, Ruth Rosenholtz , Nickson Fong, and Ulrich Neumann. Visual IDs: automatic distinctive icons for desktop interfaces. In Proceedings of the 2004 SIGGRAPH Conference**,** ACM Transactions on Graphics (TOG)

[8] Martin F. Porter. An algorithm for suffix stripping . Program , volume 14.

[9] J. S. Shapiro, J. Vanderburgh, E. Northup, and D. Chizmadia. Design of the EROS Trusted Window System. In Proceedings of the 13th USENIX Security Symposium,2004.

[10] Jinwook Seo and Ben Shneiderman. A rank-by-feature framework for unsupervised,2008.

[11] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A secure and reliable bootstrap architecture," in Proceedings of the IEEE Symposium on Research in Security and Privacy. Oakland, CA: IEEE Computer Society, May 1997. [12]Andres, N.Tolia, M. Satyanarayanan, "VM Mindependent graphics acceleration," in VEE '07: Proceedings of the 3rd International Conference on Virtual Execution Environments. ACM Press, 2007

[12] C. Smowton, "Secure 3D graphics for virtual machines," in EuroSEC '09: Proceedings of the Second European Workshop on System Security. ACM, 2009

[13] E. B. Fernandez, T. Sorgente , and M. M. Larrondo-Petrie, "Even more patterns for secure operating systems," in PLoP '06: Proceedings of the 2006 Conference on Pattern Languages of Programs. New York, NY, USA: ACM, 2006